第42卷 第3期
2022年6月
桂林电子科技大学学报
Journal of Guilin University of Electronic Technology
Vol. 42, No. 3
Jun. 2022

基于晶格 Boltzmann 方法的 CUDA 加速优化

张乾毅，韦华健，赫轶男，李华兵

(桂林电子科技大学 材料科学与工程学院，广西 桂林 541004)

摘 要：为提高流体的计算效率并保证结果的准确性，利用 CUDA 编程平台和 GPU 强大的浮点计算能力，实现了基于晶格玻尔兹曼方法的泊松流模拟计算加速。设计了线性寻址和下标寻址 2 种不同寻址方式，将这 2 种寻址方式分别应用到晶格玻尔兹曼程序的格点碰撞、迁徙流动、宏观量计算等步骤中，并探讨 2 种寻址方式对程序计算效率带来的影响。同时在程序中使用统一内存管理，通过这样的方式开辟内存的变量可在主机端和设备端同时使用，简化了代码复杂度，同时降低了频繁为变量开辟内存带来的消耗。使用 Intel(R) Xeon(R) E-52620 v4 CPU，Nvidia Quadro GP100 GPU 进行计算，在线性寻址方法和下标寻址方法中分别获得了 71 倍和 25 倍 CPU 串行代码的加速比。

关键词：CUDA；晶格玻尔兹曼方法；平面泊松流；线性寻址；下标寻址

中图分类号：O414.2 **文献标志码**：A **文章编号**：1673-808X(2022)03-0240-05

CUDA accelerated optimization based on lattice Boltzmann method

ZHANG Qianyi, WEI Huajian, HE Yinan, LI Huabing

(School of Material Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: In order to improve the efficiency of fluid calculations and ensure the accuracy of the results, the CUDA programming platform and the powerful floating-point computing capabilities of the GPU are used to accelerate the Poisseuille flow simulation calculation based on the lattice Boltzmann method. Two different addressing methods, linear addressing and subscript addressing are designed, these two addressing methods are respectively applied to the lattice point collision, migration flow, and macroscopic calculation of the lattice Boltzmann program, then discuss the influence of two addressing methods on the calculation efficiency of the program. At the same time, unified memory management is used in the program, and the variables opened up in this way can be used on the host side and the device side at the same time, which simplifies the code complexity and reduces the consumption of frequently opening up memory for variables. Using Intel(R) Xeon(R) E-52620 v4 CPU and Nvidia Quadro GP100GPU for calculations, the linear addressing method and the subscript addressing method have obtained 71 times and 25 times the speedup ratio of CPU serial code respectively.

Key words: CUDA; lattice boltzmann method; plane poiseuile flow; linear addressing; subscript addressing

随着处理器时钟频率极限及“功耗墙”等问题的出现，单核解决方案被摒弃，基于多核架构的图像处理单元(GPU)逐渐出现在人们的视野中。2007 年，英伟达公司为 GPU 增加了一个易用的编程接口——统一计算架构(compute unified device architecture，简称 CUDA)[1]。它以标准 C 为基础进行扩展，使程序员可以在 C、C++及 Fortran 等开发环境进行并行程序编写，将利用 GPU 进行并行计算的门槛大大降低。近年来，人工智能在深度学习的不断推动下高速发展，这离不开 GPU 的强大计算能力和优秀的 GPU 编程环境。CUDA 技术已经获得广泛关注，并被应用到流体力学、生物计算、气象分析、金融分析等众多领域。

晶格玻尔兹曼方法(LBM)是一种介观尺度的模拟方法，在流体力学方面的模拟中被广泛应用，如多孔介质流[2]、血液流[3]、多相流[4]淋巴流[5]等。LBM

收稿日期：2020-12-23
基金项目：国家自然科学基金(11362005)
通信作者：李华兵(1972—)，男，教授，博士，研究方向为计算物理。E-mail：hbli@guet.edu.cn
引文格式：张乾毅，韦华健，赫轶男，等. 基于晶格 Boltzmann 方法的 CUDA 加速优化[J]. 桂林电子科技大学学报，2022，42(3)：240-244.



物理背景清晰,易于并行计算,边界条件处理简单,因而十分适用于大规模 GPU 并行计算。如 Tölke 等[6]使用 CUDA 实现了 LBM 多孔介质流动模型的计算加速。郑彦奎等[7]实现了 LBM 方腔模型的计算加速。

鉴于此,通过线性寻址和下标寻址方法实现了 LBM 模型中的泊松流算例。将程序分别用 CPU 和 GPU 进行计算,比较 2 种寻址方法分别在 2 种处理单元上的计算时间,并算得加速比。

1　GPU 与 CUDA

基于设计目标的差异,GPU 在设备架构上与 CPU 存在很大区别。CPU 需要很强的通用性来处理各种不同的数据类型,同时逻辑判断又会引入大量的分支跳转和中断处理。因此,CPU 中分布着多样的计算、控制单元和占据大量空间的存储单元。而 GPU 的设计主要是用来解决大量逻辑上相对简单的任务,因而 GPU 采用了数量众多的核心和算术逻辑单元(ALU),其数量远超 CPU,通常 CPU 的核心数不会超过 2 位数,但 GPU 只配备了简单的控制逻辑和简化了的存储单元。

CUDA 编程模型将 CPU 作为主机端,GPU 作为设备端,二者各自拥有相互独立的存储地址空间:主机端内存和设备端显存。一个典型的 CUDA 程序包括并行代码和与其互补的串行代码。由 CPU 执行复杂逻辑处理和事务处理等不适合数据并行的串行代码,并调用 GPU 计算密集型的大规模数据并行计算代码,即内核函数。当设备端开始执行内核函数时,设备中会产生大量的线程,线程是内核函数的基本单元,负责执行内核函数的指定语句[8]。CUDA 程序模型如图 1 所示,由一个内核启动所产生的所有线程统称为一个线程网格,同一网格中的所有线程共享相同的全局内存空间。一个网格又由多个线程块构成,同一线程块内的线程协作可通过同步和共享内存的方式实现,不同块内的线程不能协作。每个 GPU 设备包含一组流多处理器(SM),一个 SM 又由多个流处理器(SP)和一些其他硬件组成。CUDA 程序执行过程中会把线程块中的所有线程以线程束(Warp,含 32 个线程)为执行单位分配给 SM 中的不同 SP 来进行计算。

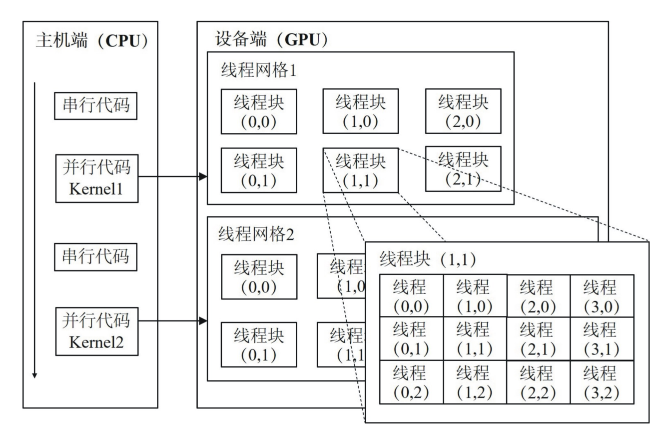


图 1　CUDA 程序模型

GPU 采用了内存分层架构,并通过多级缓存机制来提高读写效率。CUDA 内存模型如图 2 所示,GPU 中主要的内存空间包括寄存器、共享内存、全局

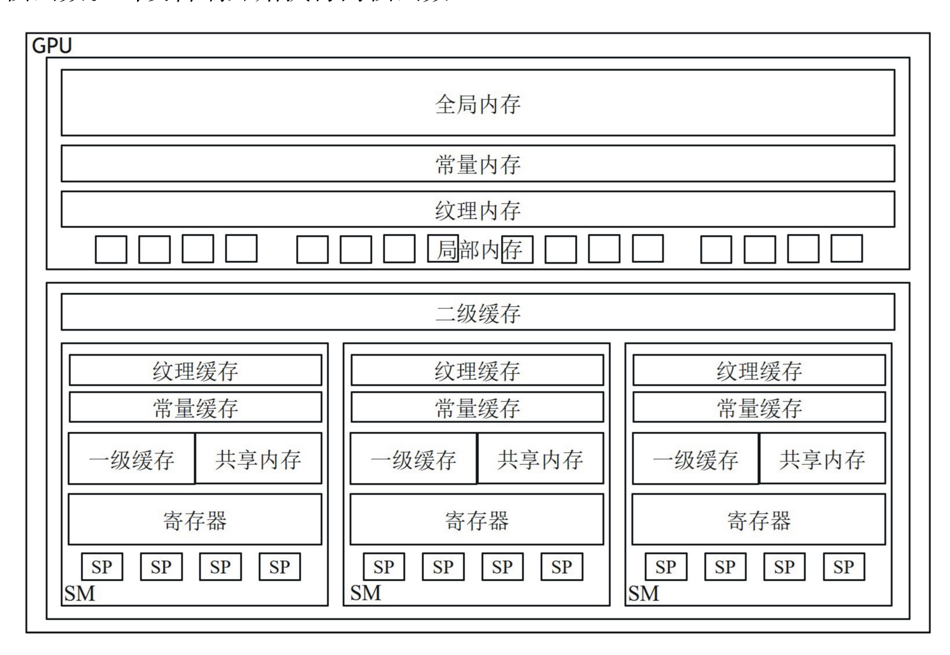


图 2　CUDA 内存模型

内存、常量内存、纹理内存和本地内存。每个 SM 都拥有独立的一级缓存，所有 SM 共享二级缓存。GPU 致力于为每个线程分配真实的寄存器，当寄存器使用到达上限时，编译器会将数据放在片外的本地内存中。共享内存是可受程序员控制的一级缓存，每个 SM 中的一级缓存与共享内存公用同一个内存段[9]，它仅对正在执行的线程块中的每个线程可见。

2 晶格玻尔兹曼方法(LBM)

McNamara 等[10]用单粒子分布函数 f_i 取代了格子气细胞自动机中的布尔变量，其 LBM 演化方程为

$$f_i(\boldsymbol{x}+\boldsymbol{e}_i\delta_t, t+\delta_t)-f_i(\boldsymbol{x},t)=\Omega(f_i), \quad (1)$$

其中：i 为微观速度的索引，对于 D2Q9 模型，i 取值 1～9；$f_i(\boldsymbol{x},t)$ 为在 $\boldsymbol{x}$ 位置 t 时刻具有 $\boldsymbol{e}_i$ 速度的粒子的分布函数；δ_t 为时间增量；$\Omega(f_i)$ 为碰撞因子，表示碰撞对于 f_i 的影响。文献[11-13]采用单弛豫时间来替代碰撞因子项。继而，LBM 方程可写为

$$f_i(\boldsymbol{x}+\boldsymbol{e}_i\delta_x, t+\delta_t)-f_i(\boldsymbol{x},t)=-\frac{1}{\tau}(f_i-f_i^{(eq)}), \quad (2)$$

其中：$f_i^{(eq)}$ 为局域平衡分布函数；τ 为弛豫时间。求出粒子分布函数后，则宏观密度、流体的流速分别为

$$\rho=\sum_i f_i=\sum_i f_i^{(eq)}, \quad (3)$$

$$u=\frac{\sum_i \boldsymbol{e}_i f_i}{\rho}=\frac{\sum_i \boldsymbol{e}_i f_i^{(eq)}}{\rho}。 \quad (4)$$

采用图 3 所示的 D2Q9 模型(D 指维度，Q 指粒子运动方向总数)模型[14]进行计算，其平衡分布函数具体定义为

$$f_i^{(eq)}=\rho\omega_i\left[1+\frac{3}{c^2}(\boldsymbol{e}_i u)+\frac{9}{2c^4}(\boldsymbol{e}_i u)^2-\frac{3}{2c^2}u^2\right], \quad (5)$$

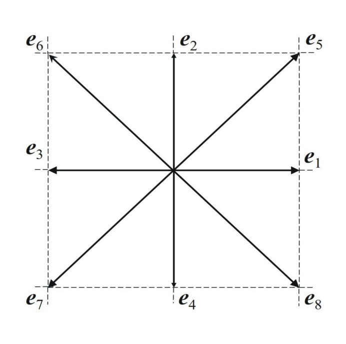


图 3 D2Q9 模型

其中：c 为基准速度，通常情况下取 1；ω_i 为各方向的权重系数。当 $i=0$ 时，$\omega_i=4/9$；当 $i=1,2,3,4$ 时，$\omega_i=1/9$；当 $i=5,6,7,8$ 时，$\omega_i=1/36$。$\boldsymbol{e}_i$ 的形式为

$$\boldsymbol{e}_i=c\begin{bmatrix}0 & 1 & 0 & -1 & 0 & 1 & -1 & -1 & 1\\ 0 & 0 & 1 & 0 & -1 & 1 & 1 & 1 & -1\end{bmatrix}。$$

3 算法实现及优化

一个典型的 LBM 计算案例主要分为 3 个步骤：1)声明变量并进行分布函数的初始化；2)进行格点的碰撞、迁徙流动和边界处理，并进行宏观量的计算；3)进行稳定性条件判断，决定是否结束步骤 2)的迭代。由文献[15]知，步骤 2)的迭代计算占整个计算时间的 98%，因此该步骤应在 GPU 上进行并行加速。本计算以淋巴管为物理背景，将淋巴管内流动的淋巴液视作等截面圆形管道内的泊松流，进行模拟计算。尝试用线性寻址和下标寻址 2 种不同的寻址方法进行计算，比较这 2 种寻址方法分别在 GPU 与 CPU 上执行计算时间的差异。

3.1 线性寻址程序设计

LBM 的基本变量是分布函数，进而由分布函数计算求得格子点的密度和 x、y 方向上的速度等宏观量。在串行程序的 D2Q9 模型中，分布函数通常被设为三维数组 df[i][j][k]并存储在主存中，i、j 表示整个线程网格中格点的坐标($0\leqslant i\leqslant$ width, $0\leqslant j\leqslant$ height；width、height 均为固定值，分别表示线程网格的宽度和高度)，k 为格点运动方向数，$k=0,1,\cdots,8$。

在设备端声明指针 double * df，调用 cudaMallocManaged( ) 函数在设备端开辟大小为 width * height * 9 * sizeof(double)的一维线性内存空间，用于存放每个格点上 9 个方向的分布函数，并将在显存上获得的内存空间首地址赋值给 df。使用 cudaMallocManaged()函数开辟的存储空间，无论是在串行代码中还是并行代码中，都可使用这块内存，因此只需定义一个指针即可在主机和设备端通用。继续开辟多个设备端内存空间，用于存放其他计算变量的数据。指针 double * dd 指向存储格子点密度的内存空间，指针 double * dvx、* dvy 分别指向存放 x 和 y 方向上速度的内存空间。

基于上述在全局内存中使用三维数组来储存分布函数及其他变量的方式，设计了 3 个在 CPU 端执行的 collide()、stream()、calculate()函数，分别代表格点的碰撞、迁徙流动和宏观量的计算等步骤，并将其命名为方案一。方案二则是将迭代计算过程放在 GPU 端并行执行，对应设计了 addKernelCollide()、addKernelCopy( )、addKernelStream( )、addKernelCalculate()四个内核函数来实现。相较于方案一中

执行各个步骤都需要嵌套 for 循环来遍历读写每个格点上的数据,方案二则通过内核函数用 GPU 映射出大量线程,同时对每个格点数据进行操作。

首先,调用 cudaMallocManaged()函数,在 GPU 端为分布函数、密度等变量开辟内存空间,再定义一个 parameter 结构体,为之后格子点迁徙流动做铺垫。对 4 个内核函数进行如下讨论:

1)碰撞步:启动 addKernelCollide<<<grid, block>>>(df,iSol,…,dev_p)内核函数,其中 grid 为线程网格中线程块数,将其设为 width;block 为每个线程块中的线程数,设为 9 * height;iSol 为用于判断格点是否位于边界上的变量;dev_p 为 parameter 结构体变量。声明变量 i、j、k 并赋值,i=blockIdx.x,j=threadIdx.y,k=threadIdx.x。在函数体内分别计算出线程访问分布函数一维数组的下标索引 id=adr(i,j,k)=k+(j+i * height) * 9 和其他变量的下标索引 ind=adr(i,j)=j+i * i * height。通过一次 if 判断,保留格子边界以外的所有格子点进行计算,每个线程利用式(2)计算对应格子点的分布函数:

df[id]=df[id]-(df[id]-
feq(k,dd[ind],dvx[ind],dvy[ind]))/Tau。

feq()为平衡分布函数,Tau 为弛豫时间。

2)流动迁徙步:启动 addKernelCopy<<<grid, block>>(df,dfbak)内核函数,声明变量并赋值,计算出线程访问的下标索引,方法同上。将 df[id]中的数据传递给 dfbak[id]。启动 addKernelStream<<<grid,block>>>(df,dfbak,iSol,…)内核函数,将 dfbak[id]的值赋给 df[id],此时 dfbak[id]中

id=k+[(j-dev_Prjy[k])+(i-
dev_Prjx[k]) * height)] * 9。

3)计算宏观量步:启动 addKernelCalculate<<<grid,block>>>(df,iSol,…)内核函数,声明变量并放入共享内存中。

```
__shared__ double sdf[9];//9 个分布函数
__shared__ double ddd;//格子点密度
__shared__ double vx;//x 方向速度
__shared__ double vy;//y 方向速度
……
for (k=0;k<9;k++)
  {
   sdf[k]=df[adr(i,j,k)];
  }
```

利用 sdf[k]计算得到 ddd、vx、vy 的值,最后将共享内存中的数据传回主存中。

3.2 下标寻址程序设计

基于之前的线性寻址方法对程序做以下修改。

1)声明指针 double * df0、double ** df1、double *** df,在 GPU 上开辟 3 个内存空间,并将首地址分别赋值给 df0、df1、df,具体操作如下:

```
cudaMallocManaged(void(**)&df0,width * height * 9 * sizeof(double));
cudaMallocManaged(void(**)&df1,width * height * sizeof(double *));
for (i=0;i<width;i++)
  {
    for(j=0;j<height;j++)
      {df1[i * height+j]=&df0[(i * height+j) * 9];}
  }
cudaMallocManaged(void(**)&df,width * height * sizeof(double **));
for (i=0;i<width;i++)
  {df[i]=&df1[i * height];}
```

2)声明指针 double * dd0、double ** dd,在 GPU 上开辟 2 次内存空间,并将首地址分别赋值给 dd0、dd,具体操作如下:

```
cudaMallocManaged ( void ( ** ) &dd0, width * height * sizeof(double));
cudaMallocManaged(void(**)&dd, width * sizeof (double *));
for (i=0;i<width;i++)
  {dd[i]=&dd0[i * height];}
```

3)其他宏观量修改方法同上。

在各计算步骤中,用多维数组表示不同格点位置上的分布函数和其他宏观量,形如 df[i][j][k]、dd[i][j]、dvx[i][j]、dvy[i][j]。将方案一、二按上述内容进行修改,并分别命名为方案三、四。

4 计算结果分析

采用含有 8 个 Nvidia Quadro GP100 显卡集群的服务器完成计算,GP100 显卡拥有 3 584 个 CUDA 并行计算处理核心,处理双精度浮点数的能力为 5.2 TFlop/s,搭配 16 GiB HBM2 显存,理论带宽高达 717 GiB/s。同时服务器搭载了 Intel(R) Xeon(R) CPU E-52620 v4 处理器,核心数为 8 个,主频为 2.1 GHz,编译环境为 CUDA10.0,运行环境为 Linux Ubuntu。

以平面泊松流为算例,在保证计算结果正确性的

前提下验证 2 种寻址方法对程序计算时间的影响。图 4 为 4 种方案迭代 4 000 步时的模拟结果，4 种方案结果一致，表明了程序修改的正确性。表 1 为迭代 4 000 步时，4 种方案模拟的计算时间。从表 1 可看出，基于线性寻址方式的方案一、二，GPU 相对 CPU 的加速比约为 71 倍，而基于下标寻址方法的加速比只有约 25 倍；方案一、三都用 CPU 完成迭代计算步，方案三的计算时间减少了将近三分之二，因而使用下标寻址的方式更适合 CPU 端的迭代计算；方案二、四计算时间无太大变化，表明下标寻址对 GPU 端的并行计算无明显帮助。

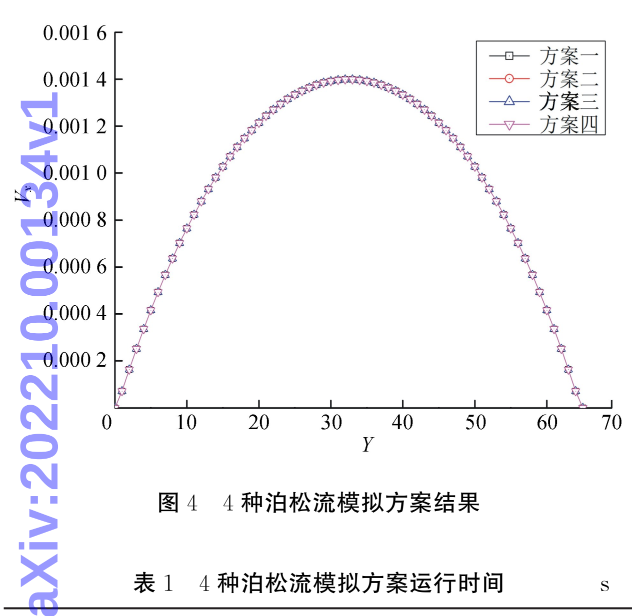


图 4　4 种泊松流模拟方案结果

表 1　4 种泊松流模拟方案运行时间　　s

方案一 CPU	方案二 GPU	方案三 CPU	方案 GPU
64.185 1	0.899 8	23.240 5	0.936 8

5　结束语

采用 CUDA 实现了 LBM 的泊松流算例。设计了线性寻址、下标寻址 2 种寻址方法，并实现了这 2 种寻址方法分别在主机端和设备端上的 4 种 LBM 计算方案。4 种方案计算结果一致，表明了程序设计的正确性。用线性寻址方法获得了 71 倍的加速比，表明用 CUDA 对 LBM 计算效率的提升有很大帮助，也展现了 GPU 在大规模科学计算中的巨大潜力。

参考文献：

[1] NVIDIA Corporation. Nvidia cuda c programming guide version 10.1[EB/OL]. (2019-11-28)[2020.11-10]. https://docs.nvidia.com/cuda/archive/10.2/cuda-c-programming-guide/index.html.

[2] 朱炼华，郭照立. 基于格子 Boltzmann 方法的多孔介质流动模拟 GPU 加速[J]. 计算物理，2015(1)：20-26.

[3] 李华兵. 晶格玻尔兹曼方法对血液流的初步研究[D]. 上海：复旦大学，2004：10-17.

[4] SHAN X，CHEN H. Lattice Boltzmann model for simulating flows with multiple phases and components[J]. Physical Review，1993，47(3)：1815-1819.

[5] MUNN L L，MEI Yumeng，BAISH J，et al. The effects of valve leaflet mechanics on lymphatic pumping assessed using numerical simulations[J]. The FASEB Journal，2020，34(1)：7043.

[6] TOLKE J，KRAFCZYK M. TeraFLOP computing on a desktop PC with GPUs for 3D CFD[J]. International Journal of Computational Fluid Dynamics，2008，22(7)：443-456.

[7] 郑彦奎，刘沙，熊生伟，等. Lattice-Boltzmann 方腔模型的 CUDA 加速实现[J]. 科学技术与工程，2010，10(7)：1684-1688.

[8] 黄昌盛，张文欢，侯志敏，等. 基于 CUDA 的格子 Boltzmann 方法：算法设计与程序优化[J]. 科学通报，2011，56(28)：2434-2444.

[9] 尚恩•库克. CUDA 并行程序设计：GPU 编程指南[M]. 苏统华，等. 译. 北京：机械工业出版社，2014：112-116.

[10] MCNAMARA G R，ZANETTI G. Use of the Boltzmann equation to simulate Lattice-Gas automata[J]. Physical Review Letters，1988，61(20)：2332.

[11] CHEN H，CHEN S，MATTHAEUS W H. Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method[J]. Physical Review a Atomic Molecular and Optical Physics，1992，45(8)：5339-5342.

[12] CHEN S，CHEN H，MARTRIAANEZ D. Lattice Boltzmann model for simulation of magnetohydrodynamics[J]. Physical Review Letters，1991，67(27)：3776-3779.

[13] QIAN Y H，D′HUMIÈRES D，PLALLEMAND P. Lattice BGK models for Navier-Stokes equation[J]. Europhysics Letters，1992，17(6)：479-484.

[14] 穆罕默德•阿卜杜勒马吉德. 格子玻尔兹曼方法：基础与工程应用[M]. 杨大勇，译. 北京：电子工业出版社，2015：20-22.

[15] 李承功. 流场的格子 Boltzmann 模拟及其 GPU-CUDA 并行计算[D]. 大连：大连理工大学，2013：47-53.

编辑：张所滨